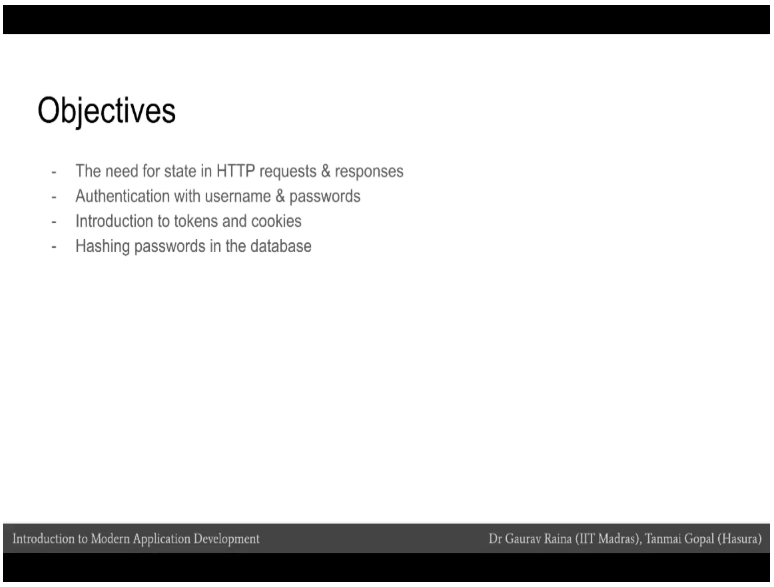
Introduction to Modern Application Development

Dr. Gaurav Raina
Prof. Tanmai Gopal
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 13
Lecture – 24
Authentication with HTTP

Hai all welcome to the first module in the performance in security unit. We are going to talk about authentication with http and which will take us into interesting discussions about how the http protocols works.

(Refer Slide Time: 00:14)



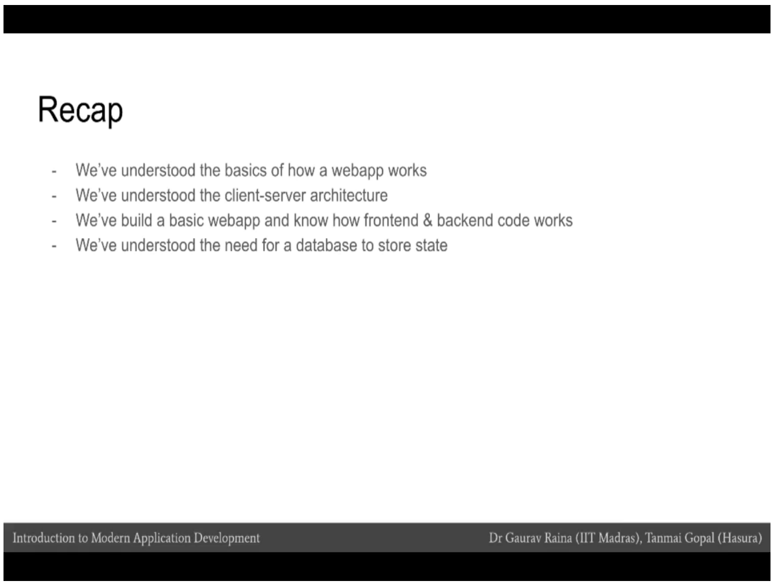
Objectives

- The need for state in HTTP requests & responses
- Authentication with username & passwords
- Introduction to tokens and cookies
- Hashing passwords in the database

Introduction to Modern Application Development   Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

Talking about usernames passwords and the concept of session tokens and cookies will also be looking at how usernames and passwords are stored in the database and why they are stored in a particular way.

(Refer Slide Time: 00:23)



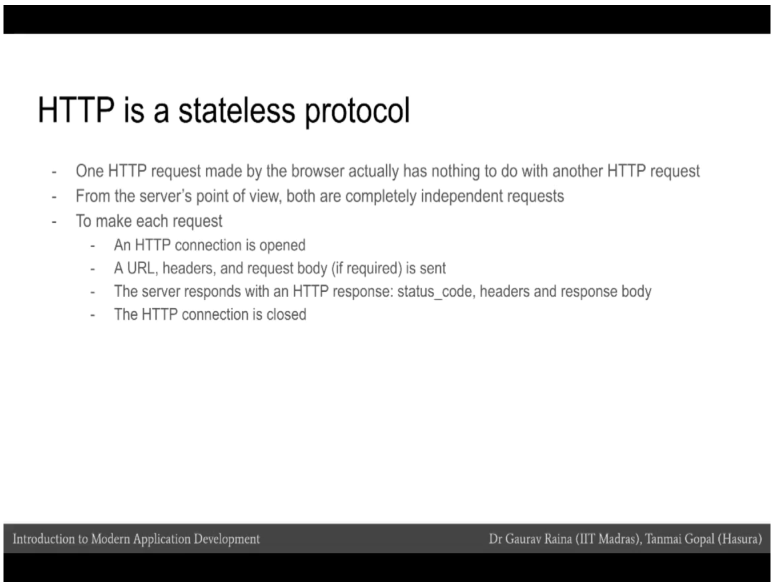
Recap

- We've understood the basics of how a webapp works
- We've understood the client-server architecture
- We've build a basic webapp and know how frontend & backend code works
- We've understood the need for a database to store state

Introduction to Modern Application Development — Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, so far over the last few units we understood the basics of how webapp works we understood what the clients server architecture looks like and what is the different constraints are. We also build a basic webapp and we understand what the front end code the back end code looks like and we also explore the database in a fair amount of depth.

(Refer Slide Time: 00:43)



HTTP is a stateless protocol

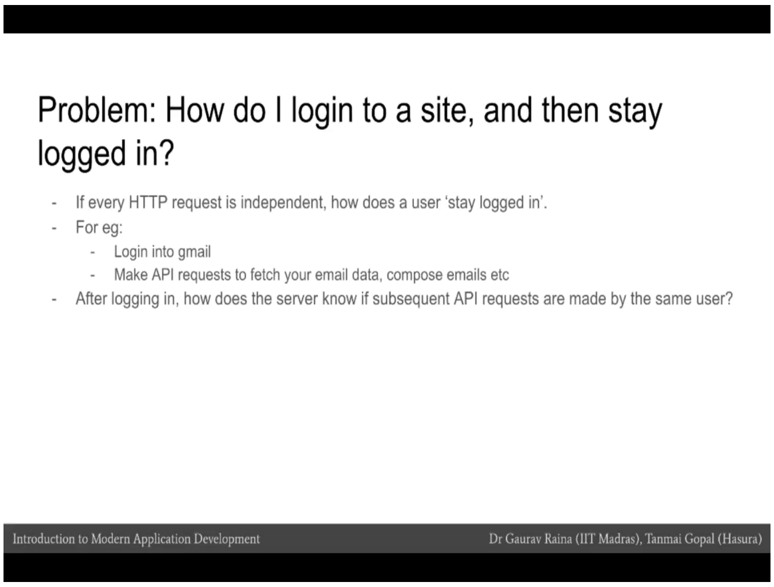
- One HTTP request made by the browser actually has nothing to do with another HTTP request
- From the server's point of view, both are completely independent requests
- To make each request
    - An HTTP connection is opened
    - A URL, headers, and request body (if required) is sent
    - The server responds with an HTTP response: status_code, headers and response body
    - The HTTP connection is closed

Introduction to Modern Application Development — Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

HTTP is what is called a stateless protocol what this means is that one http request that is made by the browser, actually has nothing to do with another http request. From the servers point of view when the server gets an http request both of the request are completely independent.

We have seen this even in the code that we wrote, in the (Refer Time: 00:59) code that we were writing we where handling each request independently. Nowhere in the request that we factoring in the fact that we knew that has somebody has made request one, we also knows that person is making request two. Whenever the request is made the browser opens up an http connection, which sense the URL, the headers and the request body, the server than response with an http response; that means, we can contain status code or contains some response headers and it or contains the response body and then the browser closes the http connection.

(Refer Slide Time: 01:31)



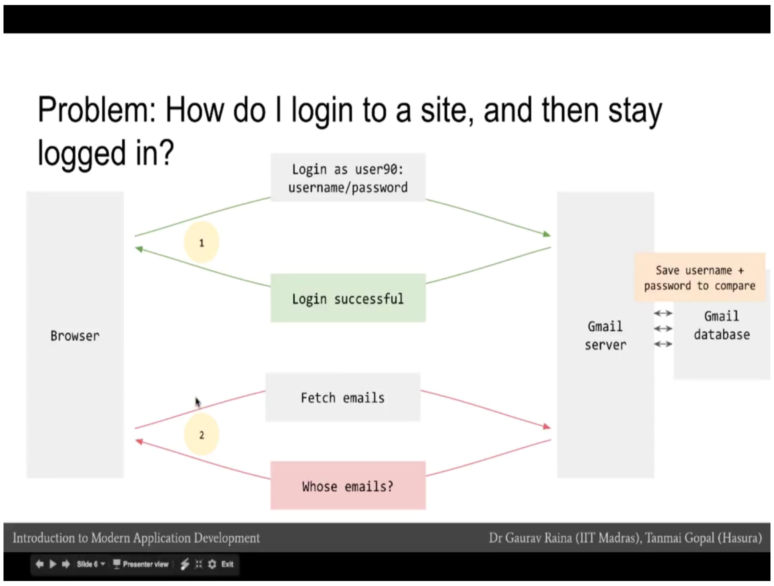
Problem: How do I login to a site, and then stay logged in?

- If every HTTP request is independent, how does a user 'stay logged in'.
- For eg:
  - Login into gmail
  - Make API requests to fetch your email data, compose emails etc
- After logging in, how does the server know if subsequent API requests are made by the same user?

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, this is one request is one cycle. If this is how http works, then how does login work? For example, if I go to Gmail and I login and then I make an API request to fetch my email data, how does that login process work what if I just directly make the API request to fetch the emails or what if I directly go to my inbox page, how does Google know or how does a Gmail server know that I am trying to make a request to the inbox page and I am not logged in (Refer Time: 01:56) director to the login page and then once I am logged in how does the Gmail server know that this is user tanmai gopal who is requesting his email data, how is that link maintain.

(Refer Slide Time: 02:07)

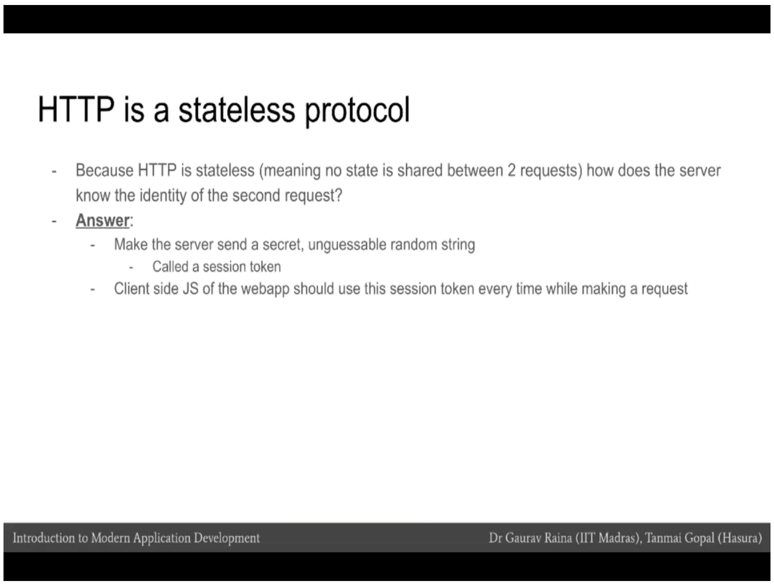


Problem: How do I login to a site, and then stay logged in?

Login as user90: username/password
Browser
1
Login successful
Gmail server
Save username + password to compare
Gmail database
Fetch emails
2
Whose emails?
Introduction to Modern Application Development
Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, just to explain this problem be more clearly on the left we have a browser, the browser first goes to the login page and makes a login request. Laughter Let say he is logging into the user name user 90 this is user 90 who is trying to login. So, the user 90 sends the username and the password in the Gmail server, the Gmail server checks with the database to see is the username password or accurate and if the password matches to the username, the Gmail server it turns a successful login response. Then it draws a sense of fetch email request, but how does the Gmail know whose email to response with, how does the Gmail server know whether this is the same person who made the request for logging in successfully right how is that link maintained.

(Refer Slide Time: 02:46)



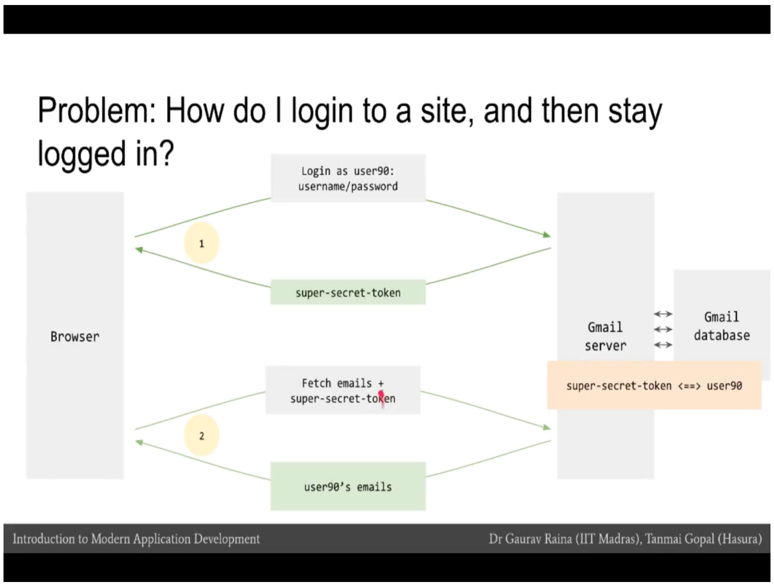
HTTP is a stateless protocol

- Because HTTP is stateless (meaning no state is shared between 2 requests) how does the server know the identity of the second request?
- **Answer**:
  - Make the server send a secret, unguessable random string
    - Called a session token
  - Client side JS of the webapp should use this session token every time while making a request

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, we only way to solve this problem is that the server should send a special string a unguessable or a random string pressably, and we can called as a session token. So, the server as to send a session token and for all subsequent request that the run accept should use the session token.

(Refer Slide Time: 03:03)



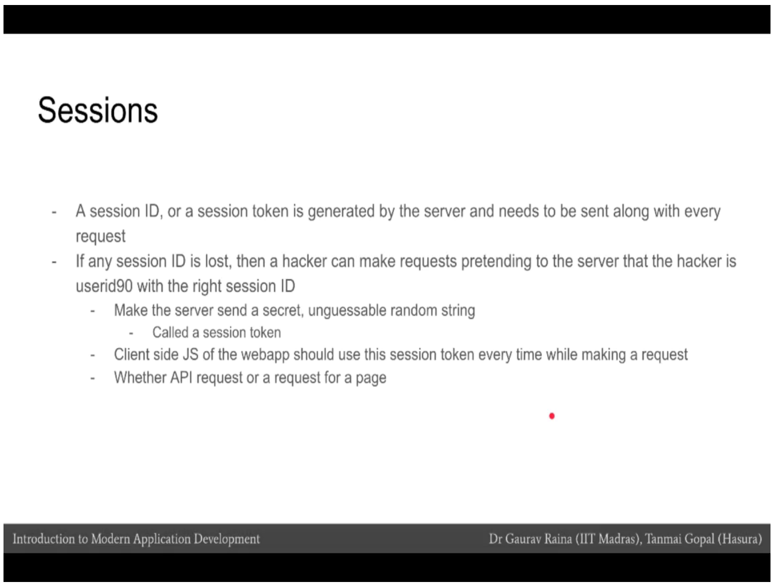
Problem: How do I login to a site, and then stay logged in?

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, let us try to see what this flow looks like; once the login is done and Gmail verifies that the login financial are correct, Gmail does not replies in the login is successful the Gmail also sends a secret token and this can be like 128 bit or a 256 bit string which is sent back to the browser. The next request for example, the request that is made to fetch the inbox (Refer Time: 03:26) fetch the emails and API, this request contains this token and when this token is sent to Gmail, Gmail as stored this token already in his database.

So, Gmail says that the token as come and this token is belongs to user 90, and because this token belongs to the user 90 Gmail server can now respond with user 90 emails right. So, this solves the problem of carrying forward information from one request response into another and it also solves the problem for authentication for Gmail to understand who is making API request right.

(Refer Slide Time: 03:57)



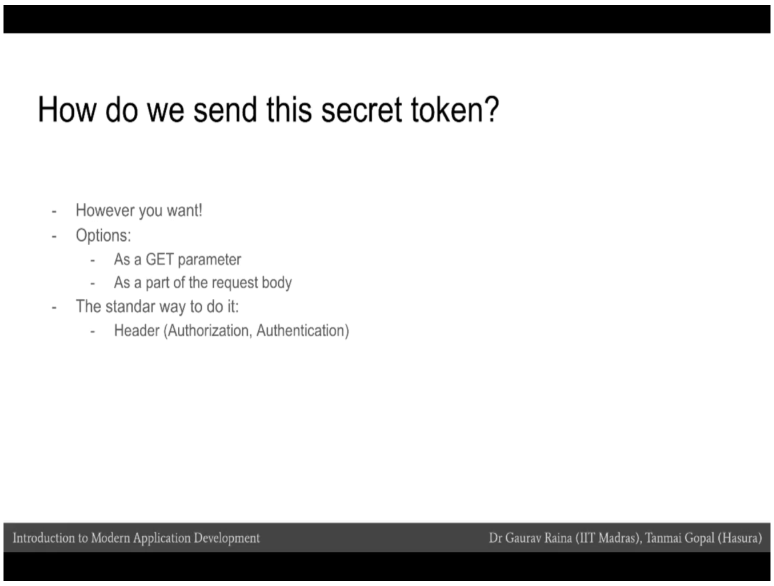
Sessions

- A session ID, or a session token is generated by the server and needs to be sent along with every request
- If any session ID is lost, then a hacker can make requests pretending to the server that the hacker is userid90 with the right session ID
  - Make the server send a secret, unguessable random string
    - Called a session token
  - Client side JS of the webapp should use this session token every time while making a request
  - Whether API request or a request for a page

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, session ID or a session token is generated by a server and it needs to be sent along with every request by the client, which an (Refer Time: 04:03) for the browser. It is important to understand that if the session ID is lost or compromised and a hacker gets access to our session ID, then the hacker can make request using that session ID which will allow the hacker to fetch the data that belongs to the user.

(Refer Slide Time: 04:36)



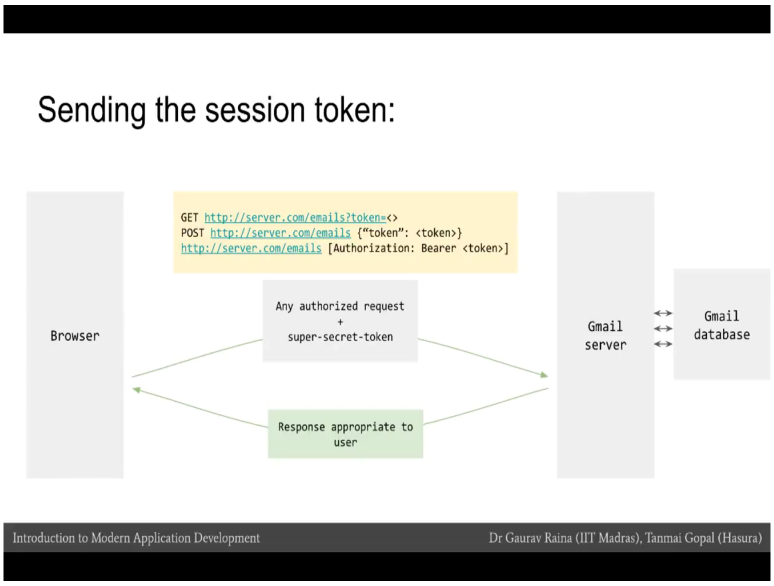
How do we send this secret token?

- However you want!
- Options:
  - As a GET parameter
  - As a part of the request body
- The standar way to do it:
  - Header (Authorization, Authentication)

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

Another important property of the session ID of the session token is that the client side javascript, must use this session token every time at makes an a js or an API request and also when the browser request access to privilege page for example, the slash email pages of the slash inbox page, then the token it also be send along. So, those page request. So, how do we send this secret token to the server for example, if we are the developers who are writing the front end or the html and the javascript or a particular application, how will be send this secret token to the server when we are making request. Because many options and in fact, we can do whatever is convenient to us. So, the c r in comes holes of both the front end code and the back end code meaning the client side code, and the server side code then we can agree and any particular contract we send that secret token. For example, we can send as a get parameter we can send it we can send the token as the part of the request body, the standard way to do it is to send it as a http header what is called a authorization header.

(Refer Slide Time: 05:11)



Sending the session token:

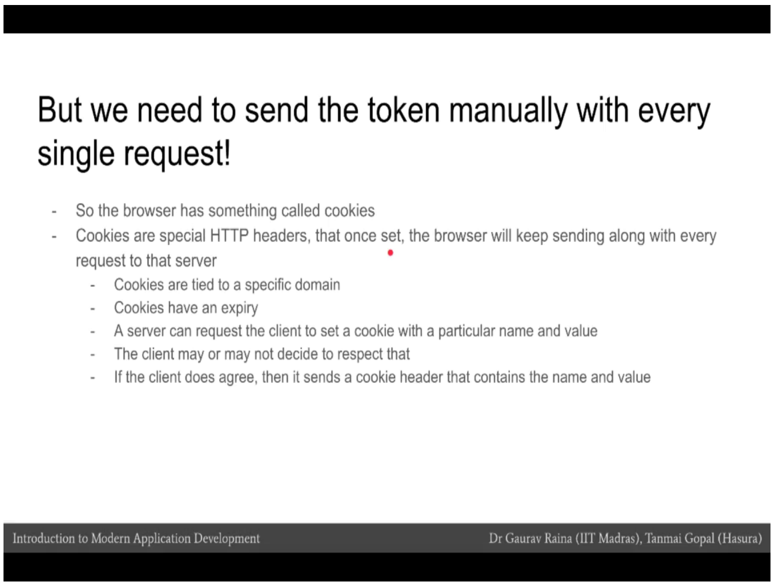
```
GET http://server.com/emails?token=<>
POST http://server.com/emails {“token”: <token>}
http://server.com/emails [Authorization: Bearer <token>]
```

Browser
Any authorized request + super-secret-token
Response appropriate to user
Gmail server
Gmail database

So, let see what this looks like when the browser is now making a request to the Gmail server it making a authorize request, it might be making an API request or it might be making a request for page. So, a many different ways what the token can be send, for example, in a get request you can say question mark token equal to so that means, the this parameter is sent and what is the Gmail server be doing is it will extract the value of this token, and check if you are access to this email page and other way to do it is to make a get or post request, but sent what is called an http header, that contains authorization and the token. So, this is called an authorization header.

In fact, even create your own http header, you can call it my secret token and that can be the name of your http header and you can send the token as a value in that. You can also make a post request on end point in this route typically be an API, in this case we can send the token even as a request body.

(Refer Slide Time: 06:03)



But we need to send the token manually with every single request!

- So the browser has something called cookies
- Cookies are special HTTP headers, that once set, the browser will keep sending along with every request to that server
    - Cookies are tied to a specific domain
    - Cookies have an expiry
    - A server can request the client to set a cookie with a particular name and value
    - The client may or may not decide to respect that
    - If the client does agree, then it sends a cookie header that contains the name and value

Introduction to Modern Application Development   Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

However the standard way to do it is to use an http header. If you think about this is actually little in convenient, every single API request that we make we would have to remember to attacks the token when we make that request, and this entire system breaks down if you want to have access to special pages.

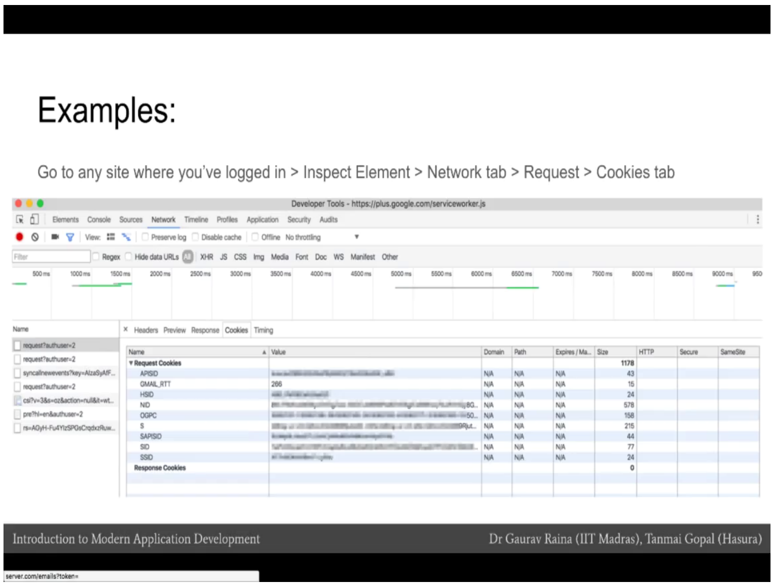
For example suppose we have a page called server dot com slash emails and this is actually a web page right. We want to have an end point that looks that looks nice for example, we want to say server dot com slash emails I do not want my end point to see to be server dot com slash emails question mark token equal to a real example for this is a Facebook profile page, when you go to a Facebook profile page you got to Facebook dot com slash username right. So, mightily should be say Facebook dot com slash (Refer Time: 06:43).

So, if you try to go to this page how can we sent a longer token so that Facebook knows that we actually have access to this page right because there is now way for us to send an extra parameter as a part of the URL, because we want a URL as look good or we want to be able to say this URLS. So, at solve this kind of a problem something called cookies exist. Cookies are just like http header, but they special http headers; these http headers are automatically managed by the browser, if a cookies has been set for a particular domain the browser takes from the responsibility of making sure the that http header is always sent to the server.

So, cookie is just another http header, but it is a special http header in the sense if the browser always attaches the http header whenever it makes a request this request can be a get request for example, it can be the link through get take or this request can be a post request which is an API request, but the browser who will take the responsibility or sending this http header. Cookies are tied to a specific domain so that means, that you can send a cookie only to a specific domain; cookies that are created by the Gmail server will not be attached by the browser for the Facebook server.

Cookies can have an expiry time. So, the browser will automatically take care of deleting that cookie and not send that as an http header after the expiry time is cross, the server can request the client to set a cookie. So, cookies can be set by the server of course, the client may or may not decide to set the cookie for example, if I making a request from the terminal where if I make the curl request or a user command by curl or a user command called w get, to make the request then cookies are not managed by this commands, but when the client is a browser, the browser understands a cookie header and actually sets the cookie.

(Refer Slide Time: 08:27)



Examples:

Go to any site where you've logged in > Inspect Element > Network tab > Request > Cookies tab

Developer Tools - https://plus.google.com/serviceworker.js

Elements Console Sources Network Timeline Profiles Application Security Audits

Preserve log Disable cache Offline No throttling

Regex Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name

- request?authuser=2
- request?authuser=2
- syncallnewevents?key=AIzaSyAfF...
- request?authuser=2
- csi?v=3&s=oz&action=null&it=wt...
- pre?hl=en&authuser=2
- rs=AGyH-Fu4YIzSPGsCrqdxzRuw...

Headers Preview Response Cookies Timing

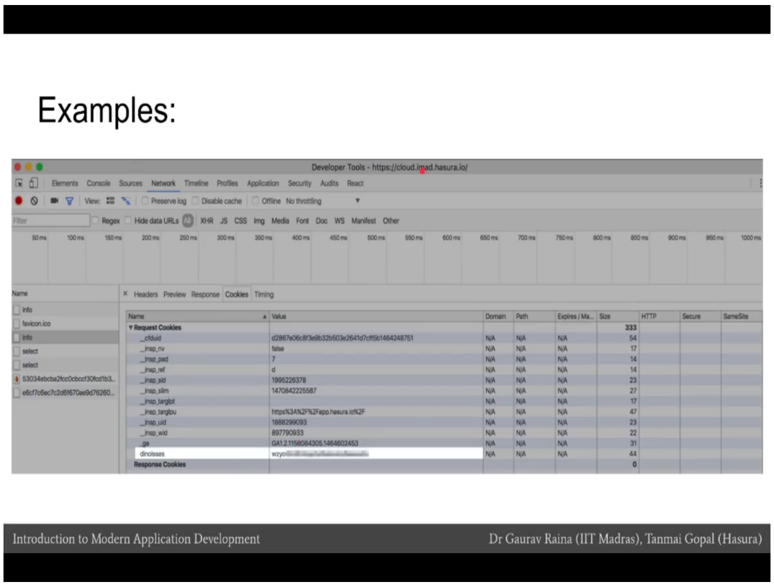
Name	Value	Domain	Path	Expires / Ma...	Size	HTTP	Secure	SameSite
▼ Request Cookies					1178			
APISID	[redacted]	N/A	N/A	N/A	43			
GMAIL_RTT	266	N/A	N/A	N/A	15			
HSID	[redacted]	N/A	N/A	N/A	24			
NID	[redacted]	N/A	N/A	N/A	578			
OGPC	[redacted]	N/A	N/A	N/A	158			
S	[redacted]	N/A	N/A	N/A	215			
SAPISID	[redacted]	N/A	N/A	N/A	44			
SID	[redacted]	N/A	N/A	N/A	77			
SSID	[redacted]	N/A	N/A	N/A	24			
Response Cookies					0			

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

server.com/emails?token=

So, to see cookies that are been used on whatever sight that you are earn, go to a web page right click open on inspect element at to the network tab, click on a particular request and then click on the cookies tab that you will see. When you click on cookie tab you will see all the different cookies that are being sent to the Gmail server whenever request is being made. So, in Gmail case we are using all other different cookies (Refer Time: 08:52) for different purposes, but the cookies that are of importance to a sub SID and SSID cookie which contain our identity.

(Refer Slide Time: 08:59)



Examples:

Developer Tools - https://cloud.imad.hasura.io/

Elements Console Sources Network Timeline Profiles Application Security Audits React

View: Preserve log Disable cache Offline No throttling

Filter Regex Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name: info, favicon.ico, info, select, select, 53034ebcba2fcc0cbccf30fcd1b3..., e6cf7c6ec7c2d6f670ae9d76260...

Headers Preview Response Cookies Timing

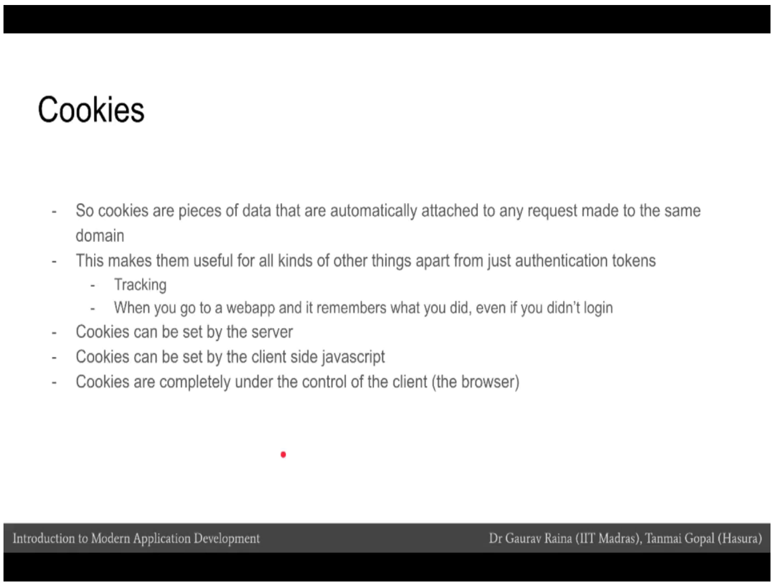
Name	Value	Domain	Path	Expires / Ma...	Size	HTTP	Secure	SameSite
▼ Request Cookies					333			
__cfduid	d2867e06c8f3e9b32b503e2641d7cff5b1464248751	N/A	N/A	N/A	54			
__insp_nv	false	N/A	N/A	N/A	17			
__insp_pad	7	N/A	N/A	N/A	14			
__insp_ref	d	N/A	N/A	N/A	14			
__insp_sid	1995226378	N/A	N/A	N/A	23			
__insp_slim	1470842225587	N/A	N/A	N/A	27			
__insp_targlpt		N/A	N/A	N/A	17			
__insp_targlpu	https%3A%2F%2Fapp.hasura.io%2F	N/A	N/A	N/A	47			
__insp_uid	1888299093	N/A	N/A	N/A	23			
__insp_wid	897790933	N/A	N/A	N/A	22			
_ga	GA1.2.1158084305.1464602453	N/A	N/A	N/A	31			
dinoisses	wzyo[redacted]	N/A	N/A	N/A	44			
Response Cookies					0			

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

If you might to the inoide console you will see that a particular cookies being used to identify who you are as user to actually handle the login.

So, for example, in this case you see this cookie and this cookie as a particular value. So, using this value the server knows that when you go to cloud that imad dot server dot io, it loads at the page that is relevant to you. This is how the inoide console works for example, if I login I will see my homepage with my database credentials my (Refer Time: 09:26) credentials and when (Refer Time: 09:27) code console it will load up my database right. Similarly if you are login you go to cloud at I am at imad dot observe io it will load your credentials right and the way this is done is because it has access to the cookie and the server uses this cookie to figure out your identity.

(Refer Slide Time: 09:42)



Cookies

- So cookies are pieces of data that are automatically attached to any request made to the same domain
- This makes them useful for all kinds of other things apart from just authentication tokens
  - Tracking
  - When you go to a webapp and it remembers what you did, even if you didn't login
- Cookies can be set by the server
- Cookies can be set by the client side javascript
- Cookies are completely under the control of the client (the browser)

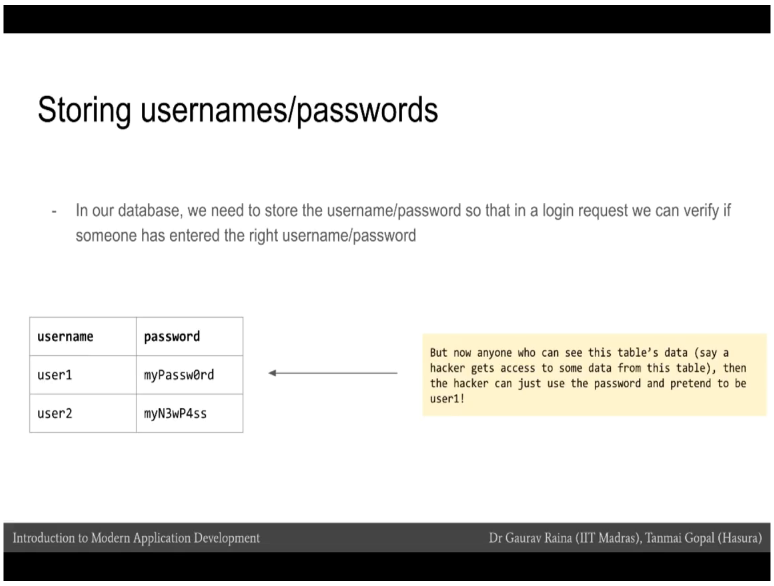
Introduction to Modern Application Development　　Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, cookies are basically pieces of data that are automatically attached to request made to the same domain. This makes cookies useful for all kinds of different things not just using them as mechanisms to exchange session information or exchange authentication token. Cookies are used most commonly cookies are used for tracking this is why you might have heard that cookies are dangerous or cookies are used for tracking and this is. In fact, how Google analytics or how most tracking servers work they have a particular cookie that they set and they whenever you visit a page the Google analytics script makes a request to the Google analytics server using that cookie and that is how Google analytics knows what are the different pages that are being visited by the same particular user. As I mentioned earlier cookies can be set by the server, but the cookie can also be set by the clients like javascript.

So, for example, we can use cookies for creating shopping cart experiences, let us say for example, you are on ecommerce sight and many e commerce sights allow you to add items to a cart without actually logging in. So, this means that whenever you add an item to a cart, the entire cart data is actually stored on a cookie which means if you shut your browser page and you e commerce sight again on the same browser, the browser will show you your cart that contains of the old cart items because the cookie is still dead right. So, we can use the cookie for storing clients side information as well very very important to understand that cookies are completely under the control of a browser.

So, if the browser is a malicious browser or the browser has a bulk then the browser might send incorrect cookies. So, the browser might not send cookies at all it is simply a convention that the browser follows in which it decides to send cookies to the server again.

(Refer Slide Time: 11:23)



Storing usernames/passwords

- In our database, we need to store the username/password so that in a login request we can verify if someone has entered the right username/password

username	password
user1	myPassw0rd
user2	myN3wP4ss

But now anyone who can see this table's data (say a hacker gets access to some data from this table), then the hacker can just use the password and pretend to be user1!

Introduction to Modern Application Development   Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

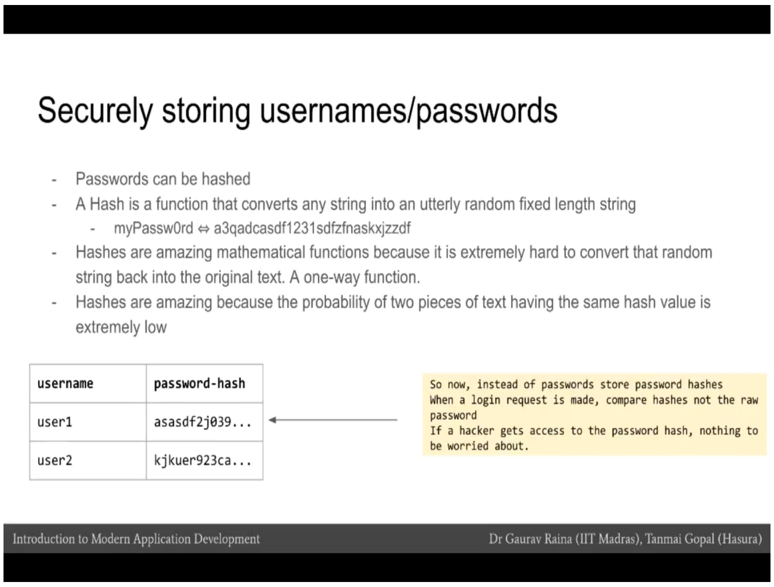
Now that we understood how session tokens work and how we can exchange session tokens to the servers let us come back to the idea of a username and a password.

When we login we send the username and the password to a server a server compares the username and password by value that it already has typically the value stored in a database. So, for example, we might have a database tables that contains two columns username and password, and let us say user one as a password called mypassword and user two as a password called my new pass. When we try to login at the back you will set the login credentials for the same username that is being given in the login and match the password values if the password values of the same then the server will response the successful login.

However this is actually a little dangerous and the reason why this is dangerous is because anybody who can get access to this table data which is unfortunately a fairly common attack then hackers that ables to get access to the database, in that case hackers now have access to be raw password that is stored by every user. So that means, the hacker can now login can pretend to be user one the hacker can also start doing actions that are controlled by the password for example, changing your password or for example, authorizing a transaction or many ecommerce sites or many banking sides you will be as to enter your password or your pin number again.

So, not only does the hacker have access to your data, but the hacker can also start making privilege transactions because the password is the value will be wrong. It might seem like there is no alternative one this is the only way to solve the problem and the only thing that we can do secure access to database, but there is actually very interesting way to try to solve this problem.

(Refer Slide Time: 12:57)



Securely storing usernames/passwords

- Passwords can be hashed
- A Hash is a function that converts any string into an utterly random fixed length string
  - myPassw0rd ⇔ a3qadcasdf1231sdfzfnaskxjzzdf
- Hashes are amazing mathematical functions because it is extremely hard to convert that random string back into the original text. A one-way function.
- Hashes are amazing because the probability of two pieces of text having the same hash value is extremely low

username	password-hash
user1	asasdf2j039...
user2	kjkuer923ca...

```
So now, instead of passwords store password hashes
When a login request is made, compare hashes not the raw
password
If a hacker gets access to the password hash, nothing to
be worried about.
```

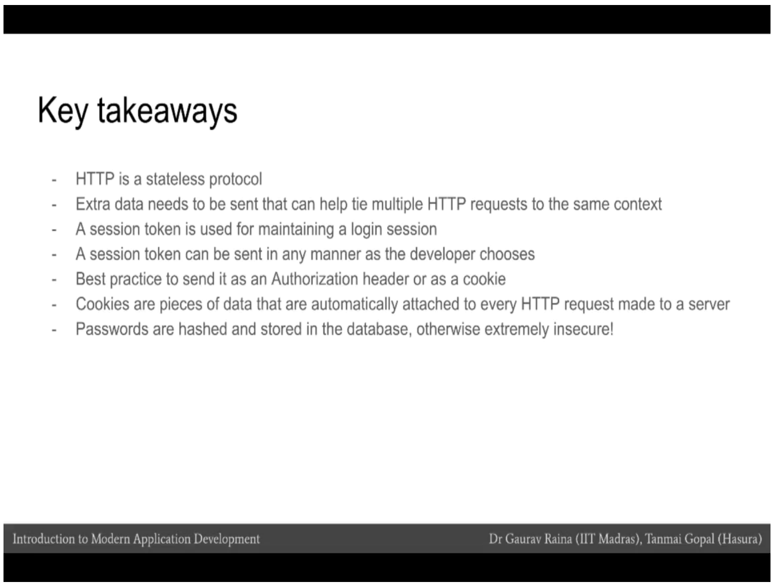
Introduction to Modern Application Development   Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

We can store passwords and we can store password like objects right securely in a database.

There is a function called a hash function and a hash function is a rather amazing mathematical function that converts any string into a fixed length randomized string and the amazing thing about the hash function is that you can go from one direction to the others. So, you can has the value of my password and converts this into as random string, but you cannot go back. That means, that if I that even if you know what the hash function is and you know the algorithm of the function that is being used and you have this random string it is almost impossible for you to recovery the original string that this hash value comes from and this is one of the most amazing mathematical inventions that has made model username password functionality possible.

So, now the way the server works is that whenever a login prediction comes with say user one and my password, the web app fetches the data from the database it fetches the data belonging to user one, it fetches this password hash the webapp takes the password that is being given in login credentials which is mypassword applies the hash algorithm on it and compares that to the password hash that it add in the database. If those two hash in match then the password will be same and if the hash is do not match then the password will not the same; this allows us to compare passwords without knowing what the password really is.

(Refer Slide Time: 14:24)



Key takeaways

- HTTP is a stateless protocol
- Extra data needs to be sent that can help tie multiple HTTP requests to the same context
- A session token is used for maintaining a login session
- A session token can be sent in any manner as the developer chooses
- Best practice to send it as an Authorization header or as a cookie
- Cookies are pieces of data that are automatically attached to every HTTP request made to a server
- Passwords are hashed and stored in the database, otherwise extremely insecure!

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, in this module we understood how http is a stateless protocol and how that means, that as developers we have to sent extra data that can help as tie multiple http request to the same context, we understood that we can use the concept of a session token to maintain a login session, we also know that the session token can be exchanged with the server in whatever format that we that we choose, we look at two popular conventions for sending this session token a one is sending it as a http header as an authorization header there is another popular convention of cookies which are used to extents of machine, we also understood that cookies are actually more useful than just sending session tokens and they can be used for doing several things like tracking or storing clients and information we also looked at the concept of hashing and how password the securely stored in a database.